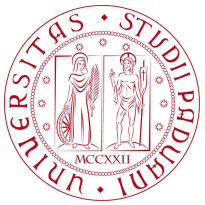
Esempi di utilizzo di generatori random delle librerie Boost ++

Alberto Garfagnini

Università degli studi di Padova

December 4, 2013



Introduzione

- Le librerie Boost ++ aggiungono molte funzionalità alle librerie standard del C++
- Una descrizione delle API (Application Programming Interface) si trova al sito http://www.boost.org
- Una buona introduzione alle librerie è fornita dalla copia elettronica del libro http://en.highscore.de/cpp/boost/index.html
- le librerie implementano un buon numero di generatori di numeri pseudo-casuali e forniscono la possibilità di campionare da varie distribuzioni di probabilità

Esempi di generatori nella libreria Boost++

Generatore	Periodo	Memory req.	Speed
`rand48`	$2^{48} - 1$	sizeof(uint64_t)	64%
`hellekalek1995`	$2^{31} - 1$	sizeof(int32_t)	2%
`mt19937`	$2^{19937} - 1$	625*sizeof(uint32_t)	93%
`lagged_fibonacci607`	$\sim 2^{32000}$	607*sizeof(double)	59%
`lagged_fibonacci19937`	$\sim 2^{1050000}$	19937*sizeof(double)	59%

- la velocità di generazione è rapportata al generatore che offre la prestazione migliore (100% = massime prestazioni)
- `hellekalek1995` fornisce una buona distribuzione uniforme in svariate dimensioni

Da
http://www.boost.org/doc/libs/1_48_0/doc/html/boost_random/reference.htm

Esempi di utilizzo di un generatore

- Per poter generare da una distribuzione di probabilità è necessario:
  1. scegliere un generatore di numeri pseudo-random (`rnd`);
  2. impostare il seme della generazione con il metodo
     `rnd.seed( t )`
  3. definire la distribuzione di probabilità dalla quale campionare
  4. creare un `variate_generator` combinando il generatore pseudo-random e la distribuzione di probabilità
- il file header `boost/random.hpp` contiene tutti gli include files necessari
- in fase di compilazione si deve indicare la directory dove trovare gli header file

```
g++ -I /usr/include/boost myrandom_fibonacci.cxx
```

Es 1 : distrib. uniforme - Gen: Mersenne-Twister

```
#include <iostream>
#include "boost/random.hpp"
using namespace std;
int main(int argc, char ** argv)
{
    // Uniform distribution [0, 1]
    boost::uniform_real<> uni(0.0, 1.0);

    // Generator: Mersenne-Twister
    boost::mt19937 rng_1;
    rng_1.seed( 1 );
    boost::variate_generator < boost::mt19937 &,
                               boost::uniform_real<> > uni_rng_1(rng_1,uni);
    if (argc<2) {
        cout << "Usage: " << argv[0] << " numeri_da_generare\n";
        return 1;
    }
    int n = strtol(argv[1], NULL, 0);

    for (int i=0; i<n; i++)
        cout << uni_rng_1() << endl;

    return 0;
}
```

Es 1 : distrib. uniforme - Gen: Lagged-Fibonacci

```
#include <iostream>
#include "boost/random.hpp"
using namespace std;
int main(int argc, char ** argv)
{
    // Select the probability distribution
    boost::uniform_real<> uni(0.0, 1.0);

    // Select the random number generator
    boost::lagged_fibonacci607 rng_2;
    rng_2.seed( 1 );
    boost::variate_generator <boost::lagged_fibonacci607 &,
                              boost::uniform_real<> > uni_rng_2(rng_2,uni);
    if (argc<2) {
        cout << "Usage: " << argv[0] << " numeri_da_generare\n";
        return 1;
    }
    int n = strtol(argv[1], NULL, 0);
    for (int i=0; i<n; i++)
        cout << uni_rng_2() << endl;

    return 0;
}
```

Es 1 : distrib. Gauss - Gen: LCG

```
#include <iostream>
#include "boost/random.hpp"
using namespace std;
int main(int argc, char ** argv)
{
    // Gaussian distrib. mean = 2.0, sigma = 0.2
    boost::normal_distribution<> norm(2.0, 0.2);

    // Linear Congruential Generator
    boost::rand48 rng;

    rng.seed( 1 );

    boost::variate_generator < boost::rand48 &,
                               boost::normal_distribution<> >
                                      norm_rng( rng, norm );
    if (argc<2) {
        cout << "Usage: " << argv[0] << " numeri_da_generare\n";
        return 1;
    }
    int n = strtol(argv[1], NULL, 0);
    for (int i=0; i<n; i++) {
        cout << norm_rng() << endl;
    }
    return 0;
}
```